

Relatório Trabalho Prático 2

Programação Dinâmica e Programação Gulosa

Bruna Gomes Camilo

Erick Henrique Dutra de Souza

Lucas Cota Dornelas

Julho de 2021

Laboratório de Algoritmo e Estrutura de Dados II

1. Descrição

O presente trabalho consiste em implementar uma solução a partir de um algoritmo para o problema das linhas de montagem, apresentado em aula, onde se quer verificar qual é o caminho mais eficiente da entrada nas linhas de montagem até a saída levando em consideração o tempo de processamento em cada estação e o tempo de transporte entre uma estação e outra, assim como o tempo de saída das linhas de montagem.

A Programação Dinâmica tem como intuito solucionar a problemática de otimização por intermédio da análise de uma sequência de problemas simplificados com relação ao problema original. Dessa maneira, determina-se uma variável e soluciona o problema que detém uma variável a menos ($N-1$) para se alcançar a resolução do problema original de N variáveis. A resolução do problema de $N-1$ variáveis é dada em função da resolução de um problema de $N-2$ variáveis e assim por diante. Portanto, a programação dinâmica consiste em realizar a solução de todos os possíveis subproblemas de um problema maior, iniciando dos menores e terminando nos menores, assim, armazena-se os resultados encontrados em uma matriz.

Por outro lado, o algoritmo guloso soluciona uma problemática fazendo a escolha que parece ser a melhor no momento agora. Desse modo, o algoritmo guloso realiza uma escolha local ótima e almeja que essa seja também uma solução globalmente ótima. Em virtude do método utilizado, o algoritmo guloso ignora uma grande parte de caminhos possíveis que, por conseguinte, o permite permear por um conjunto relativamente enxuto de soluções.

A comparação entre os dois métodos leva-nos a algumas diferenças. Com relação ao método de resolução na programação dinâmica a escolha depende da solução de subproblemas, já o algoritmo guloso escolhe a melhor solução no momento. A solução do problema na programação dinâmica é feita de baixo para cima, isto é, se parte dos problemas menores para os maiores, por outro lado, no algoritmo guloso é feito de cima para baixo. Por fim, na programação dinâmica os

resultados dos subprogramas são salvo facilitando possíveis correções o que não acontece com o algoritmo guloso.

2. Implementação


Para implementarmos o algoritmo, utilizamos previamente a classe `Dinamica` do pacote `dinamica`, para a implementação do algoritmo de Programação Dinâmica e Guloso, `XAEDsMaps` e `Dijkstra` do pacote `guloso`, contida no livro 'Projeto de Algoritmos com implementações em Java e C++', do autor Nivio Ziviani, como base para a implementação do algoritmo. Além disso, temos as classes `FPHeapMinIndireto`, `Grafo` e `LinhaDeMontagem` do pacote `util`, que auxilia com alguns métodos as classes definidas nos pacotes previamente estabelecidos.

Além disso, temos a classe `Main` do pacote `main` que cria uma novas linhas de montagem, que nos permite implementar os algoritmos desenvolvidos, bem como executá-los para os casos de testes.

Sendo assim, temos:

pacote `dinamica`

classe `Dinamica`

```
dinamica >  Dinamica.java > ...  
Lucas Cota, an hour ago | 2 authors (You and others)  
1 package dinamica;  
2  
3 import util.LinhaDeMontagem;  
4  
Lucas Cota, an hour ago | 2 authors (You and others)  
5 public class Dinamica {  
6  
7     private LinhaDeMontagem L1, L2;  
8     private int[] tempoMin1, tempoMin2;  
9     private int[] caminho1, caminho2; // vetores para guardar o caminho de quais linhas devem ser seguidas  
10    private int tempoFinal, linhaFinal; // valores que armazenarao a saida otima  
11    private int i;
```

dinamica > Dinamica.java > ...

```
12
13 public Dinamica(LinhaDeMontagem L1, LinhaDeMontagem L2) {
14     this.tempoMin1 = new int[L1.getStationTime().length - 2];
15     this.tempoMin2 = new int[L2.getStationTime().length - 2];
16     this.caminho1 = new int[L1.getStationTime().length - 2];
17     this.caminho2 = new int[L2.getStationTime().length - 2];
18
19     // inicializa a primeira posicao do primeiro vetor dos caminhos minimos
20     tempoMin1[0] = L1.getStationTime()[0] + L1.getStationTime()[1];
21     tempoMin2[0] = L2.getStationTime()[0] + L2.getStationTime()[1];
22
23     caminho1[0] = 1;
24     caminho2[0] = 2;
25
26     this.L1 = L1;
27     this.L2 = L2;
28     this.i = 1;
29 }
30
31 // método que calcula o caminho minimo entre a entrada e saida
32 public void caminhoMinimo() {
33     // loop que começa do 2 para que nao haja conflitos em arrays diferentes e de tamanhos diferentes
34     for (i = 2; i <= L1.getStationTime().length - 2; i++) {
35
36         // tempo para se manter na estacao 1 e 2
37         int tempo1 = tempoMin1[i - 2] + L1.getStationTime()[i];
38         int tempo2 = tempoMin2[i - 2] + L2.getTransportTime()[i - 2] + L1.getStationTime()[i];
39
40         // compara qual dos dois custos e melhor para ser armazenado no vetor
41         if (tempo1 <= tempo2) {
42             // salva o tempo 1 que é a subsolucao otima e que o caminha contem a solução
43             tempoMin1[i - 1] = tempo1;
44             caminho1[i - 1] = 1;
45         } else {
46             // salva o tempo 2 que é a subsolucao otima e que o caminha contem a solução
47             tempoMin1[i - 1] = tempo2;
48             caminho1[i - 1] = 2;
49         }
50
51         // tempo para se manter na estacao 1 e 2
52         tempo1 = tempoMin2[i - 2] + L2.getStationTime()[i];
53         tempo2 = tempoMin1[i - 2] + L1.getTransportTime()[i - 2] + L2.getStationTime()[i];
54
55         // compara qual dos dois custos e melhor para ser armazenado no vetor
56         if (tempo1 <= tempo2) {
57             // salva o tempo 1 que é a subsolucao otima e que o caminha contem a solução
58             tempoMin2[i - 1] = tempo1;
59             caminho2[i - 1] = 2;
60         } else {
```

dinamica > Dinamica.java > Dinamica > Dinamica(LinhaDeMontagem, LinhaDeMontagem)

```
60 } else {
61     // salva o tempo 2 que é a subsolucao otima e que o caminha contem a solução
62     tempoMin2[i - 1] = tempo2;
63     caminho2[i - 1] = 1;
64 }
65 }
```

dinamica > Dinamica.java > Dinamica > Dinamica(LinhaDeMontagem, LinhaDeMontagem)

```
65     }
66
67     // calcula qual saida é otima
68     if (tempoMin1[i - 2] + L1.getStationTime()[L1.getStationTime().length - 1] <= tempoMin2[i - 2]
69         + L2.getStationTime()[L2.getStationTime().length - 1]) {
70         // armazena a saida otima caso seja a primeira e registra que e na linha 1
71         tempoFinal = tempoMin1[i - 2] + L1.getStationTime()[L1.getStationTime().length - 1];
72         linhaFinal = 1;
73     } else {
74         // armazena a saida otima caso seja a primeira e registra que e na linha 2
75         tempoFinal = tempoMin2[i - 2] + L2.getStationTime()[L2.getStationTime().length - 1];
76         linhaFinal = 2;
77     }
78
79 }
80
81 public void imprimeCaminhoMinimo() {
82     int j = linhaFinal;
83     // printa a primeira linha antes do loop
84     System.out.println("Linha: " + j + " Estação: " + (L1.getStationTime().length - 2));
85     // loop decrescente, da ultima estacao ate a primeira
86     for (i = L1.getStationTime().length - 2; i > 1; i--) {
87         // condicao para qual linha deve ser printada, de acordo com os valores salvos
88         // nos vetores l1 e l2, na funcao caminhoMinimo()
89         if (j == 1) {
90             j = caminho1[i - 1];
91         } else {
92             j = caminho2[i - 1];
93         }
94         // printa as estacoes e linhas de acordo com os valores previamente armazenados
95         System.out.println("Linha: " + j + " Estacao: " + (i - 1));
96     }
97     // ao final imprime o tempo total
98     System.out.println("\nTempo gasto: " + tempoFinal + "\n\n");
99 }
100 }
```

pacote guloso

classe Dijkstra

guloso > Dijkstra.java > ...

You, 2 hours ago | 1 author (You)

```
1  /*      You, 2 hours ago • WIP: Algorithm
2  Disponibilizado em Projeto de Algoritmos com implementações em Java e C++
3  Autor: Nivio Ziviani
4  */
5
6  You, 2 hours ago
7  package guloso;
8
9  import util.FPHeapMinIndireto;
10 import util.Grafo;
```

You, 2 hours ago | 1 author (You)

```
10 public class Dijkstra {
11
12     private int antecessor[];
13     private double p[];
14     private Grafo grafo;
15     private int tempoTotal;
16
17     public void getTempoTotal() {
18         System.out.println("\nTempo total: " + tempoTotal);
19     }
20
21     public Dijkstra(Grafo grafo) {
22         this.grafo = grafo;
23         this.tempoTotal = 0;
24     }
25
26     public void obterArvoreCMC(int raiz) throws Exception {
27         int n = this.grafo.numVertices();
28         this.p = new double[n]; //peso dos vertices
29         int vs[] = new int[n + 1]; //vertices
30         this.antecessor = new int[n];
31         for (int u = 0; u < n; u++) {
32             this.antecessor[u] = -1;
33             p[u] = Double.MAX_VALUE; //p tem o maximo valor possivel
34             vs[u + 1] = u; //Heap indireto a ser construido
35         }
36         p[raiz] = 0;
37         FPHeapMinIndireto heap = new FPHeapMinIndireto(p, vs);
38         heap.constroi();
39         while (!heap.vazio()) {
40             int u = heap.retiraMin();
41             if (!this.grafo.listaAdjVazia(u)) {
42                 Grafo.Aresta adj = grafo.primeiroListaAdj(u);
43                 while (adj != null) {
44                     int v = adj.v2();
45                     if (this.p[v] > (this.p[u] + adj.peso().getPesoTotal())) {
46                         antecessor[v] = u;
47                         heap.diminuiChave(v, this.p[u] + adj.peso().getPesoTotal());
48                     }
49                     adj = grafo.proxAdj(u);
50                 }
51             }
52         }
53     }
54 }
```

```

guloso > Dijkstra.java > ...
55     public int antecessor(int u) {
56         return this.antecessor[u];
57     }
58
59     public double peso(int u) {
60         return this.p[u];
61     }
62
63     public void imprimeCaminho(int origem, int v) {
64         if (origem == v)
65             return;
66         else if (this.antecessor[v] == -1)
67             System.out.println("Nao existe caminho de " + origem + " ate " + v);
68         else {
69             imprimeCaminho(origem, this.antecessor[v]); //imprimindo recursivamente o caminho
70             System.out.println("Aresta " + antecessor[v] + " a " + v); //arestas
71             System.out.println("Distancia: " + grafo.mat[antecessor[v]][v].getDistancia() + " Tempo: " + grafo.mat[antecessor[v]][v].getDistancia()); //distancia e tempo
72             tempoTotal += grafo.mat[antecessor[v]][v].getDistancia(); //tempo total gasto
73         }
74     }
75 }
76

```

classe Guloso

```

guloso > Guloso.java > ...
You, seconds ago | 2 authors (You and others)
1  package guloso;
2
3  import util.Grafo;
4  import util.LinhaDeMontagem;
5
You, seconds ago | 2 authors (You and others)
6  public class Guloso {
7
8      private Grafo grafo;
9      private LinhaDeMontagem L1, L2;
10
11     public Guloso(LinhaDeMontagem L1, LinhaDeMontagem L2) {
12         this.L1 = L1;
13         this.L2 = L1;
14     }
15
16     public void gulosoMontagem() {
17         this.grafo = new Grafo(this.L1.getStationTime().length + this.L2.getStationTime().length - 2); //inicializando o grafo
18         grafo.insereAresta(0, 1, this.L1.getStationTime()[0] + this.L1.getStationTime()[1]);
19
20         for (int i = 1; i < this.L1.getStationTime().length - 2; i++)
21             grafo.insereAresta(i, i + 1, this.L1.getStationTime()[i + 1]);
22
23         //trabalhando com a primeira linha
24         grafo.insereAresta(this.L1.getStationTime().length - 2, grafo.numVertices() - 1, this.L1.getStationTime()[this.L1.getStationTime().length - 1]);
25         //trabalhando com a segunda linha
26         grafo.insereAresta(0, this.L1.getStationTime().length - 1, this.L2.getStationTime()[0] + this.L2.getStationTime()[1]);
27
28         for (int i = 0; i < this.L2.getStationTime().length - 2; i++)
29             //transportando entre as linhas de montagem
30             grafo.insereAresta(i + this.L1.getStationTime().length - 1, i + this.L1.getStationTime().length, this.L2.getStationTime()[i + 2]);
31
32         for (int i = 0; i < this.L1.getTransportTime().length; i++) {
33             grafo.insereAresta(i + 1, i + this.L1.getStationTime().length, this.L1.getTransportTime()[i] + this.L2.getStationTime()[i + 2]);
34             grafo.insereAresta(i + this.L1.getStationTime().length - 1, i + 2, this.L2.getTransportTime()[i] + this.L1.getStationTime()[i + 2]);
35         }
36     }
37
38     public void caminhoMinimoGuloso() throws Exception {
39         XAEDsMaps x = new XAEDsMaps();
40         x.caminhoMinimoDijkstra(grafo, 0, grafo.numVertices() - 1);
41     }
42 }

```

classe XAEDsMaps

```
guloso > XAEDsMaps.java > ...  
You, an hour ago | 2 authors (You and others)  
1 package guloso;  
2  
3 import util.Grafo;  
4  
You, an hour ago | 2 authors (You and others)  
5 public class XAEDsMaps {  
6  
7     public XAEDsMaps() {  
8     }  
9  
10    public void caminhoMinimoDijkstra(Grafo grafo, int v1, int v2) throws Exception {  
11        Dijkstra d = new Dijkstra(grafo);  
12        d.obterArvoreCMC(v1);           //caminhos minimos do grafo  
13        d.imprimeCaminho(v1, v2);      //imprime o caminho minimo  
14        d.getTempoTotal();             //retorna o tempo total  
15    }  
16 }  
17
```

pacote util

classe FPHeapMinIndireto

```
util > FPHeapMinIndireto.java > FPHeapMinIndireto  
You, 2 hours ago | 2 authors (You and others)  
1 /*  
2     Disponibilizado em Projeto de Algoritmos com implementações em Java e C++  
3     Autor: Nivio Ziviani  
4 */  
You, 3 hours ago  
5 package util;  
Bruna Gomes, 2 hours ago | 2 authors (You and others)  
6 public class FPHeapMinIndireto {  
7  
8     private double p[];  
9     private int n, pos[], fp[];  
10
```


util > 1 FPHeapMinIndireto.java > FPHeapMinIndireto

```
8 private double p[];
9 private int n, pos[], fp[];
10
11 public FPHeapMinIndireto(double p[], int v[]) {
12     this.p = p;
13     this.fp = v;
14     this.n = this.fp.length - 1;
15     this.pos = new int[this.n];
16     for (int u = 0; u < this.n; u++)
17         this.pos[u] = u + 1;
18 }
19
20 public void refaz(int esq, int dir) {
21     int j = esq * 2;
22     int x = this.fp[esq];
23     while (j <= dir) {
24         if ((j < dir) && (this.p[fp[j]] > this.p[fp[j + 1]]))
25             j++;
26         if (this.p[x] <= this.p[fp[j]])
27             break;
28         this.fp[esq] = this.fp[j];
29         this.pos[fp[j]] = esq;
30         esq = j;
31         j = esq * 2;
32     }
33     this.fp[esq] = x;
34     this.pos[x] = esq;
35 }
36
37 public void constroi() {
38     int esq = n / 2 + 1;
39     while (esq > 1) {
40         esq--;
41         this.refaz(esq, this.n);
42     }
43 }
44
45 You, 3 hours ago • WIP: Algorithm
46
47 public int retiraMin() throws Exception {
48     int minimo;
49     if (this.n < 1)
50         throw new Exception("Erro: heap vazio");
51     else {
52         minimo = this.fp[1];
53         this.fp[1] = this.fp[this.n];
54         this.pos[fp[this.n--]] = 1;
55         this.refaz(1, this.n);
56     }
57     return minimo;
58 }
```

util > FPHeapMinIndireto.java > FPHeapMinIndireto

```
56     }
57
58     public void diminuiChave(int i, double chaveNova) throws Exception {
59         i = this.pos[i];
60         int x = fp[i];
61         if (chaveNova < 0)
62             throw new Exception("Erro: chaveNova com valor incorreto");
63         this.p[x] = chaveNova;
64         while ((i > 1) && (this.p[x] <= this.p[fp[i / 2]])) {
65             this.fp[i] = this.fp[i / 2];
66             this.pos[fp[i / 2]] = i;
67             i /= 2;
68         }
69         this.fp[i] = x;
70         this.pos[x] = i;
71     }
72
73     public boolean vazio() {
74         return this.n == 0;
75     }
76
77     public void imprime() {
78         for (int i = 1; i <= this.n; i++)
79             System.out.print(this.p[fp[i]] + " ");
80         System.out.println();
81     }
82 }
```

classe Grafo

util > Grafo.java > Grafo > Peso

You, 2 hours ago | 2 authors (You and others)

```
1  /*
2   |  Disponibilizado em Projeto de Algoritmos com implementações em Java e C++
3   |  Autor: Nivio Ziviani
4   */
   You, 3 hours ago
5  package util;
```

util > Grafo.java > Grafo > Peso

You, 3 hours ago

```
5 package util;
```

```
6
```

You, 2 hours ago | 2 authors (You and others)

```
7 public class Grafo {
```

You, 2 hours ago | 1 author (You)

```
8     public class Peso {
```

```
9         private int distancia, tempo;
```

```
10
```

```
11         public Peso(int distancia, int tempo) {
```

```
12             this.distancia = distancia;
```

```
13             this.tempo = tempo;
```

```
14         }
```

```
15
```

```
16         public Peso(int distancia) {
```

```
17             this.distancia = distancia;
```

```
18             this.tempo = 0;
```

```
19         }
```

```
20
```

```
21         public int getDistancia() {
```

```
22             return this.distancia;
```

```
23         }
```

```
24
```

```
25         public int getTempo() {
```

```
26             return this.tempo;
```

```
27         }
```

```
28
```

```
29         public void setDistancia(int distancia) {
```

```
30             this.distancia = distancia;
```

```
31         }
```

```
32
```

```
33         public void setTempo(int tempo) {
```

```
34             this.tempo = tempo;
```

```
35         }
```

```
36
```

```
37         public int getPesoTotal() {
```

```
38             return this.distancia + this.tempo;
```

```
39         }
```

```
40     } // You, 3 hours ago • WIP: Algorithm
```

```
41     @Override
```

```
42     public String toString() {
```

```
43         return distancia + "";
```

```
44     }
```

```
45
```

```
46
```

util > Grafo.java > Grafo > Peso

You, 3 hours ago | 1 author (You)

```
47 public static class Aresta {
48
49     private int v1, v2;
50     private Peso peso;
51
52     public Aresta(int v1, int v2, Peso peso) {
53         this.v1 = v1;
54         this.v2 = v2;
55         this.peso = peso;
56     }
57
58     public Peso peso() {
59         return this.peso;
60     }
61
62     public int v1() {
63         return this.v1;
64     }
65
66     public int v2() {
67         return this.v2;
68     }
69
70     public void setV1(int v1) {
71         this.v1 = v1;
72     }
73
74     public void setV2(int v2) {
75         this.v2 = v2;
76     }
77
78     public void setPeso(Peso peso) {
79         this.peso = peso;
80     }
81
82 }
83
84 public Peso mat[][]; // pesos do tipo inteiro
85 private int numVertices;
86 private int pos[]; // posicao atual ao se percorrer os adjs de um vertice v
87
88 public Grafo(int numVertices) {
```

util > Grafo.java > Grafo > Peso

```
88 public Grafo(int numVertices) {
89     this.mat = new Peso[numVertices][numVertices];
90     this.pos = new int[numVertices];
91     this.numVertices = numVertices;
92     for (int i = 0; i < this.numVertices; i++) {
93         for (int j = 0; j < this.numVertices; j++)
94             this.mat[i][j] = null;
95         this.pos[i] = -1;
96     }
97 }
```

```
98
99     public void insereAresta(int v1, int v2, int distancia, int tempo) {
100         Peso p = new Peso(distancia, tempo);
101         this.mat[v1][v2] = p;
102     }
103
104     public void insereAresta(int v1, int v2, int distancia) {
105         Peso p = new Peso(distancia);
106         this.mat[v1][v2] = p;
107     }
108
109     public boolean existeAresta(int v1, int v2) {
110         return (this.mat[v1][v2] != null);
111     }
112
113     public boolean listaAdjVazia(int v) {
114         for (int i = 0; i < this.numVertices; i++) {
115             if (this.mat[v][i] != null)
116                 return false;
117         }
118         return true;
119     }
120
121     public Aresta primeiroListaAdj(int v) {
122         this.pos[v] = -1;
123         return this.proxAdj(v);
124     }
125
126     public Aresta proxAdj(int v) {
127         this.pos[v]++;
128         while ((this.pos[v] < this.numVertices) && (this.mat[v][this.pos[v]] == null))
129             this.pos[v]++;
130         if (this.pos[v] == this.numVertices)
131             return null;
132         else
133             return new Aresta(v, this.pos[v], this.mat[v][this.pos[v]]);
134     }
135
136
137     public Aresta retiraAresta(int v1, int v2) {
138         if (this.mat[v1][v2] == null)
139             return null; //Aresta nao existe
140         else {
141             Aresta aresta = new Aresta(v1, v2, this.mat[v1][v2]);
142             this.mat[v1][v2] = null;
143             return aresta;
144         }
145     }
```

util > Grafo.java > Grafo > Peso

```
147     public int numVertices() {
148         return this.numVertices;
149     }
150
151     public Grafo grafoTransposto() {
152         Grafo grafoT = new Grafo(this.numVertices);
153         for (int v = 0; v < this.numVertices; v++) {
154             if (!this.listaAdjVazia(v)) {
155                 Aresta adj = this.primeiroListaAdj(v);
156                 while (adj != null) {
157                     grafoT.insereAresta(adj.v2(), adj.v1(), adj.peso().distancia, adj.peso().tempo);
158                     adj = this.proxAdj(v);
159                 }
160             }
161         }
162         return grafoT;
163     }
164
165     public void imprime() {
166         System.out.println();
167         for (int i = 0; i < this.numVertices; i++) {
168             System.out.print "[" + i + "]-> ");
169             for (int j = 0; j < this.numVertices; j++) {
170                 if (this.mat[i][j] != null)
171                     System.out.print(j + " > ");
172             }
173             System.out.println(".");
174         }
175         System.out.println();
176     }
177 }
178
```

classe LinhaDeMontagem

```

util > LinhaDeMontagem.java > ...
1  package util;
2
3  public class LinhaDeMontagem {
4
5      private int[] stationTime;
6      private int[] transportTime;
7
8      public LinhaDeMontagem(int[] stationTime, int[] transportTime) {
9          this.stationTime = stationTime;
10         this.transportTime = transportTime;
11     }
12
13     public int[] getStationTime() {
14         return stationTime;
15     }
16
17     public int[] getTransportTime() {
18         return transportTime;
19     }
20 }

```

pacote main

classe Main

```

main > Main.java > Main > main(String[])
Lucas Cota, an hour ago | 2 authors (You and others)
1  package main;
2
3  import util.LinhaDeMontagem;
4  import dinamica.Dinamica;
5  import guloso.*;
6
7  public class Main {
8      public static void main(String[] args) throws Exception {
9          Guloso guloso;
10         Dinamica programacaoDinamica;
11         LinhaDeMontagem L1;
12         LinhaDeMontagem L2;

```

main > Main.java > Main > main(String[])

```
14      System.out.println("Instancia - 1");
15
16      // Situação do problema 1
17      int[] A1 = new int[] {3,5,7,10,5,9,11,9,5,2,6};
18      int[] A2 = new int[] {2,6,3,9,11,4,9,3,12,4,5};
19
20      int[] T1 = new int[] {3,5,4,2,7,5,8,1};
21      int[] T2 = new int[] {5,3,7,5,6,2,5,2};
22
23      L1 = new LinhaDeMontagem(A1, T1);
24      L2 = new LinhaDeMontagem(A2, T2);
25
26      System.out.println("\nGuloso: ");
27
28      // Aplicando o algoritmo guloso para a primeira instancia
29      guloso = new Guloso(L1, L2);
30      guloso.gulosoMontagem();
31      guloso.caminhoMinimoGuloso();
32
33      System.out.println("\nProgramacao Dinamica: \n");
34
35      programacaoDinamica = new Dinamica(L1, L2);
36      programacaoDinamica.caminhoMinimo();
37      programacaoDinamica.imprimeCaminhoMinimo();
38
39      System.out.println("Instancia - 2");
40
41      // Situação do problema 2
42      A1 = new int[] {5,10,6,3,8,5,3,7,12,8};
43      A2 = new int[] {7,3,5,3,7,6,4,9,10,9};
44      T1 = new int[] {4,2,7,2,5,8,2};
45      T2 = new int[] {6,1,7,3,6,4,5};
46
47      L1 = new LinhaDeMontagem(A1, T1); //criando a linha 1
48      L2 = new LinhaDeMontagem(A2, T2); //criando a linha 2
49
50      System.out.println("\nGuloso: ");
51
52      guloso = new Guloso(L1, L2);
53      guloso.gulosoMontagem();
54      guloso.caminhoMinimoGuloso();
55
56      System.out.println("\nProgramacao Dinamica: \n");
57
58      programacaoDinamica = new Dinamica(L1, L2);
59      programacaoDinamica.caminhoMinimo();
60      programacaoDinamica.imprimeCaminhoMinimo();
61
```

You, 2 hours ago • FI

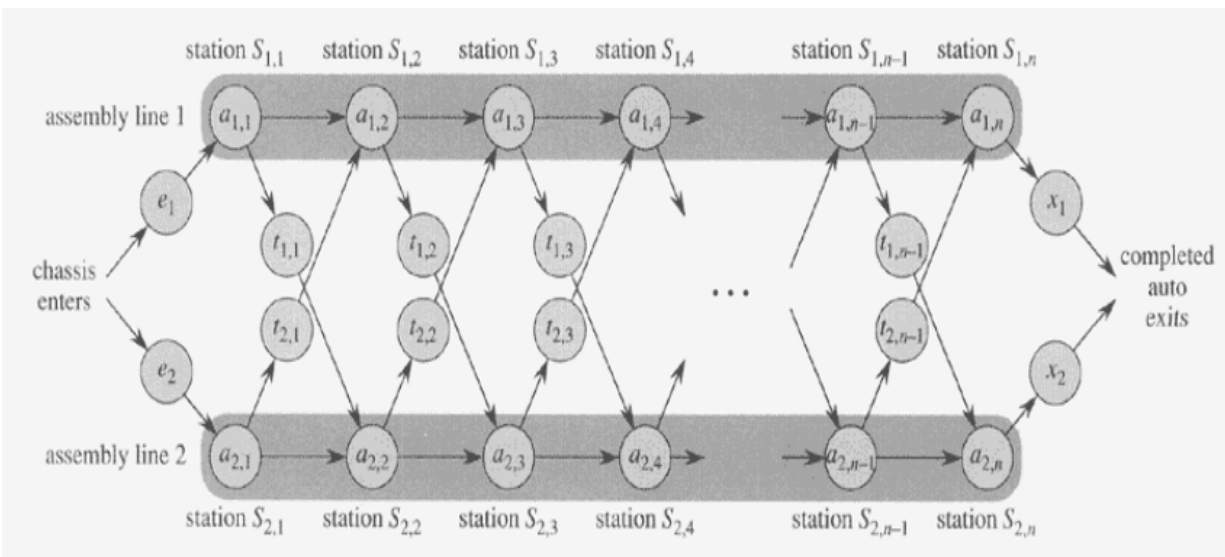

```

56     System.out.println("\nProgramacao Dinamica: \n");
57
58     programacaoDinamica = new Dinamica(L1, L2);
59     programacaoDinamica.caminhoMinimo();
60     programacaoDinamica.imprimeCaminhoMinimo();
61 }
62

```

3. Casos de teste

Temos como base, o seguinte problema:



Para executar o algoritmo desenvolvido, temos os seguintes casos de testes que serão executados para o algoritmo de Programação Dinâmica e Programação Gulosa:

TESTE 1:	A1	[3,5,7,10,5,9,11,9,5,2,6]
	A2	[2,6,3,9,11,4,9,3,12,4,5]
	T1	[3,5,4,2,7,5,8,1]
	T2	[5,3,7,5,6,2,5,2]
TESTE 2:	A1	[5,10,6,3,8,5,3,7,12,8]
	A2	[7,3,5,3,7,6,4,9,10,9]
	T1	[4,2,7,2,5,8,2]
	T2	[6,1,7,3,6,4,5]

4. Resultados

Os resultados obtidos são listados a seguir:

Instância 1							
Programação Dinâmica				Programação Gulosa			
Linha:	1	Estação:	9	Aresta 0 a 10	Distância:	8	Tempo: 8
Linha:	1	Estação:	8	Aresta 10 a 11	Distância:	7	Tempo: 7
Linha:	2	Estação:	7	Aresta 11 a 12	Distância:	10	Tempo: 10
Linha:	2	Estação:	6	Aresta 12 a 13	Distância:	5	Tempo: 5
Linha:	2	Estação:	5	Aresta 13 a 14	Distância:	9	Tempo: 9
Linha:	2	Estação:	4	Aresta 14 a 15	Distância:	11	Tempo: 11
Linha:	2	Estação:	3	Aresta 15 a 16	Distância:	9	Tempo: 9
Linha:	2	Estação:	2	Aresta 16 a 17	Distância:	5	Tempo: 5
Linha:	2	Estação:	1	Aresta 17 a 18	Distância:	2	Tempo: 2
Tempo Gasto:			65	Aresta 18 a 19	Distância:	6	Tempo: 6
				Tempo Total:			72

				Instância 2				
Programação Dinâmica				Programação Gulosa				
Linha:	1	Estação:	8	Aresta 0 a 1	Distância:	15	Tempo:	15
Linha:	1	Estação:	7	Aresta 1 a 2	Distância:	6	Tempo:	6
Linha:	1	Estação:	6	Aresta 2 a 3	Distância:	3	Tempo:	3
Linha:	1	Estação:	5	Aresta 3 a 4	Distância:	8	Tempo:	8
Linha:	1	Estação:	4	Aresta 4 a 5	Distância:	5	Tempo:	5
Linha:	1	Estação:	3	Aresta 5 a 6	Distância:	3	Tempo:	3
Linha:	2	Estação:	2	Aresta 6 a 7	Distância:	7	Tempo:	7
Linha:	2	Estação:	1	Aresta 7 a 8	Distância:	12	Tempo:	12
Tempo Gasto:			62	Aresta 8 a 17	Distância:	8	Tempo:	8
				Tempo Total:			67	

5. Considerações Finais

Ao fim da realização deste presente trabalho, conseguimos compreender não apenas de modo teórico, mas também prático os conceitos da programação dinâmica e do algoritmo guloso. Além disso, concluímos que os casos de testes foram executados com sucesso.